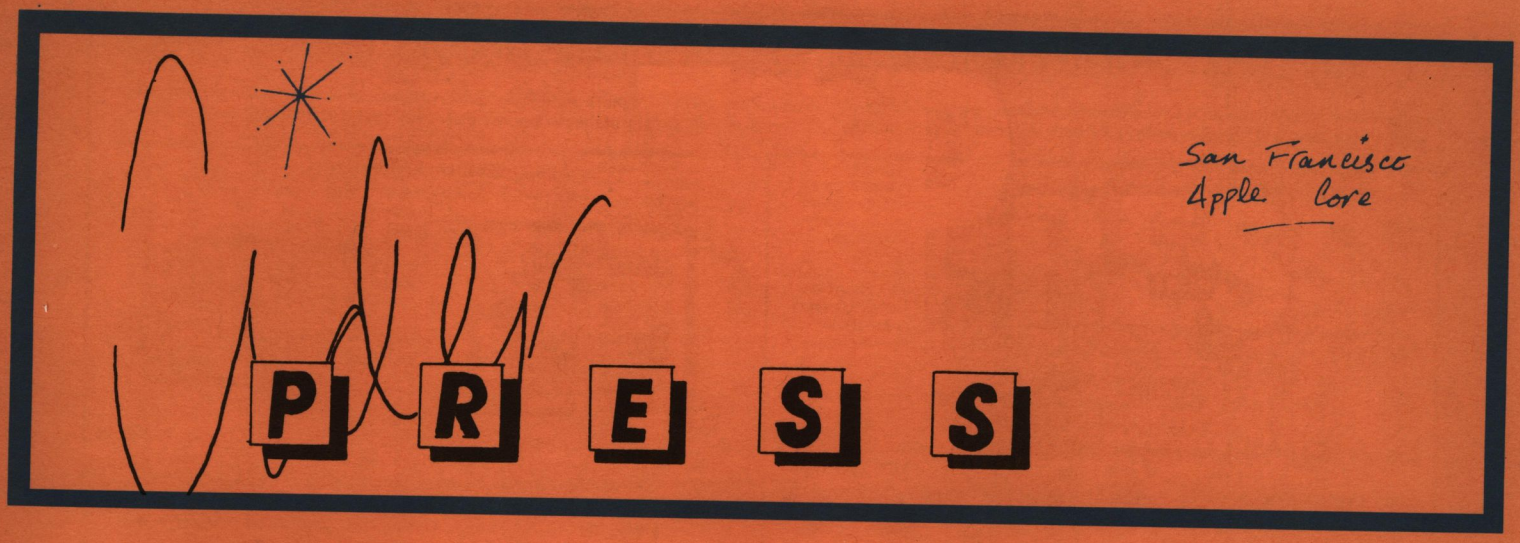


San Francisco Apple Core

IN THIS ISSUE:

Board Nominations . . . . . Joyce Conklin
B.A.S.I.C. . . . . Joyce Conklin
Pascal Easy I/O . . . . . Michael Fischer
Forth World . . . . . Ed Bronstein
File Manager--Mach. Lang. Interface . . . . . Bryan Costales
Think Tank- Review . . . . . Michael Fischer
Nuers . . . . . Kathy Rudovsky
Order Form . . . . . .

DEC 83:JAN 84

GENERAL MEETINGS:

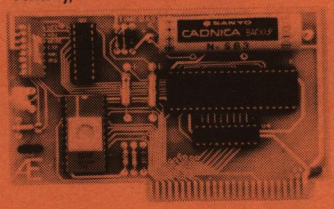
JAN 21
FEB 18

APPLIED ENGINEERING Quality Starts On The Drawing Board....

But it doesn't end there. Our commitment to excellence applies all the way down the line, through manufacturing and testing to our three year no-hassle warranty. The aim, of course, is to provide you with the best designed and best crafted hardware available. Try any of the peripherals below and see what it's like to own the finest.

The TIMEMASTER
Finally, a clock that does it ALL!

Automatically date stamps files with PRO-DOS



Fully PRO-DOS compatible

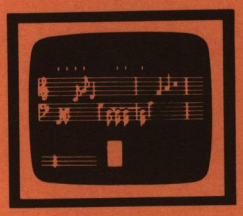
- Just plug it in and your programs can read the year, month, date, day, and time to 1 millisecond! The only clock with both year and ms.
- A rechargeable NiCad battery will keep the TIMEMASTER running for over ten years.
- Powerful 2K ROM driver — No clock could be easier to use.
- Full emulation of most other clocks, including Mountain Hardware's Appleclock (but you'll like the TIMEMASTER mode better).
- Basic, Machine Code, CP/M and Pascal software on 2 disks!
- Eight software controlled interrupts so you can execute two programs at the same time (many examples are included).
- On-board timer lets you time any interval up to 48 days long down to the nearest millisecond.

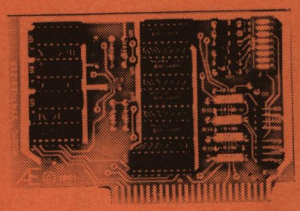
The TIMEMASTER includes 2 disks with some really fantastic time oriented programs (over 25) plus DOS dater so it will automatically add the date when disk files are created or modified. The disk is over a $200.00 value alone — we give the software others sell. All software packages for business, data base management and communications are made to read the TIMEMASTER.

If you want the most powerful and the easiest to use clock for your Apple, you want a TIMEMASTER.

PRICE $129.00

Super Music Synthesizer
Improved Hardware and Software

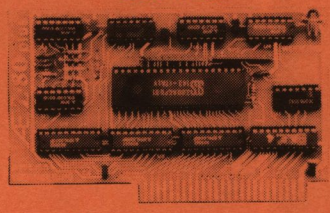




- Complete 16 voice music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo, boot the disk supplied and you are ready to input and play songs.
- It's easy to program music with our compose software. You will start right away at inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.
- Now with new improved software for the easiest and the fastest music input system available anywhere.
- We give you lots of software. In addition to Compose and Play programs, 2 disks are filled with over 30 songs ready to play.
- Easy to program in Basic to generate complex sound effects. Now your games can have explosions, phaser zaps, train whistles, death cries. You name it, this card can do it.
- Four white noise generators which are great for sound effects.
- Plays music in true stereo as well as true discrete quadraphonic.
- Full control of attack, volume, decay, sustain and release.
- Will play songs written for ALF synthesizer (ALF software will not take advantage of all our card's features. Their software sounds the same in our synthesizer.)
- Our card will play notes from 30HZ to beyond human hearing.
- Automatic shutoff on power-up or if reset is pushed.
- Many many more features.

PRICE $159.00

Z-80 PLUS!



- TOTALLY compatible with ALL CP/M software.
- The only Z-80 card with a special 2K "CP/M detector" chip.
- Fully compatible with microsoft disks (no pre-boot required).
- Specifically designed for high speed operation in the Apple IIe (runs just as fast in the II+ and Franklin).
- Red "CP/M WORKING" LED indicator, the Z-80 Plus does not interfere with non-CP/M programs.
- A semi-custom I.C. and a low parts count allows the Z-80 Plus to fly thru CP/M programs at a very low power level. (We use the Z-80A at fast 4MHZ.)
- Does EVERYTHING the other Z-80 boards do, plus Z-80 interrupts.

Don't confuse the Z-80 Plus with crude copies of the microsoft card. The Z-80 Plus employs a much more sophisticated and reliable design. With the Z-80 Plus you can access the largest body of software in existence. Two computers in one and the advantages of both, all at an unbelievably low price.

PRICE $139.00

Viewmaster 80

There used to be about a dozen 80 column cards for the Apple, now there's only ONE.

- TOTALLY Videx Compatible.
- 80 characters by 24 lines, with a sharp 7x9 dot matrix.
- On-board 40/80 soft video switch with manual 40 column override
- Fully compatible with ALL Apple languages and software—there are NO exceptions.
- Low power consumption through the use of CMOS devices.
- All connections on the card are made with standard video connectors.
- All new design (using a new Microprocessor based C.R.T. controller) for a beautiful razor sharp display.
- The VIEWMASTER incorporates all the features of all other 80 column cards, plus many new improvements.

	PRICE	BUILT IN SOFTSWITCH	SHIFT KEY SUPPORT	LOW POWER DESIGN	THREE YEAR WARRANTY	7x9 DOT MATRIX	LIGHT PEN INPUTS	40 COLUMN OVERRIDE	INVERSE CHARACTERS
VIEWMASTER	**169**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
SUP'RTERM	MORE	NO	YES	NO	NO	NO	NO	YES	YES
WIZARD80	MORE	NO	NO	NO	NO	YES	NO	YES	YES
VISION80	MORE	YES	YES	NO	NO	YES	NO	NO	NO
OMNIVISION	MORE	NO	YES	NO	NO	NO	NO	YES	YES
VIEWMAX80	MORE	YES	YES	NO	NO	YES	NO	NO	YES
SMARTERM	MORE	YES	YES	NO	NO	NO	YES	YES	NO
VIDEOTERM	MORE	NO	NO	YES	NO	YES	YES	NO	YES

The VIEWMASTER 80 works with all 80 column applications including CP/M, Pascal, WordStar, Format II, Easywriter, Apple Writer II, VisiCalc, and all others. The VIEWMASTER 80 is THE MOST compatible 80 column card you can buy at ANY price!

PRICE $169.00

MemoryMaster IIe 128K RAM Card

- Expands your Apple IIe to 192K memory.
- Provides an 80 column text display.
- Compatible with all Apple IIe 80 column and extended 80 column card software (same physical size as Apple's 64K card).
- Can be used as a solid state disk drive to make your programs run up to 20 times FASTER (the 64K configuration will act as half a drive).
- Permits your IIe to use the new double high resolution graphics.
- Automatically expands Visicalc to 95 K storage in 80 columns! The 64K config. is all that's needed, 128K can take you even higher.
- PRO-DOS will use the MemoryMaster IIe as a high speed disk drive.
- Lowest power consuming 128K card AVAILABLE.
- Complete documentation included, we show you how to use all 128K.

If you already have Apple's 64K card, just order the MEMORYMASTER IIe with 64K and use the 64K from your old board to give you a full 128K. (The board is fully socketed so you simply plug in more chips.)

MemoryMaster IIe with 128K	**$249**
Upgradeable MemoryMaster IIe with 64K	**$169**
Non-Upgradeable MemoryMaster IIe with 64K	**$149**

Our boards are far superior to most of the consumer electronics made today. All I.C.'s are in high quality sockets with mil-spec. components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products work in the APPLE IIE, II, II+ and Franklin. The MemoryMaster IIe is IIe only. Applied Engineering also manufactures a full line of data acquisition and control products for the Apple; A/D converters and digital I/O cards, etc. Please call for more information. All our products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no hassle **THREE YEAR WARRANTY.**

All Orders Shipped Same Day
Texas Residents Add 5% Sales Tax
Add $10.00 If Outside U.S.A.
Dealer Inquiries Welcome

Send Check or Money Order to:
APPLIED ENGINEERING
P.O. Box 470301
Dallas, TX 75247

Call (214) 492-2027
7a.m. to 11p.m. 7 days a week
MasterCard, Visa & C.O.D. Welcome
No extra charge for credit cards

CIDER PRESS

NOMINATIONS FOR BOARD OF DIRECTORS

At the January and February general meetings, nominations will be accepted for the Board of Directors of the San Francisco Apple Core. Following the February meeting, the ballot and any candidates' statements will be mailed to all members of record.

The ballot should be marked and returned by the date shown in the mailing so that the results of the election may be announced at the March general meeting. Members of the Board of Directors meet monthly to deal with the policy decisions necessary for the operation of the SFAC.

If you have ideas about directions which you would like to see the Core take, get yourself nominated and run!

If you think you can help the Core remain a viable force in your life and become such in the lives of others, run for a seat on the Board of Directors!

If you want to repay just some of the many hours contributed by the volunteers, run for a seat on the Board of Directors, or just plain volunteer to help!

THE CORE'S CORE

DIRECTORS:

Sami Capron	415/388-8631
Joyce Conklin	415/341-6955
Michael Fischer	415/933-4435
Gary Knox	415/521-0603
Don Scellato	415/672-2051
Alan L Smerican	415/566-2342
Allan B. Woods	415/687-9659

MEMBER SERVICE/COMMITTEES:

Membership		
Alan L Smerican		415/566-2342
	Ted Chatterton	
Cider Press -		
Editor	Allan Woods	415/687-9659
BASIC	Joyce Conklin	415/341-6955
Int. Grp. Coord.	Sami Capron	415/388-8631
Marketing Dir.	Don Scellato	415/672-2051
Review Group	("Woody") Liswood	415/585-6375
General Meeting	Michael Fischer	415/933-4435
Treasurer	Gary Knox	415/521-0603
Coordinating Dir.	Joyce Conklin	415/341-6955

CIDER PRESS POLICY

All manuscripts, photographs, and other materials will be considered submitted free and released for publication. They become the property of the San Francisco Apple Core and the CIDER PRESS. Authors should clearly mark all material submitted for publication so that proper credit can be given.

The views expressed in any articles are those of the authors, and not necessarily those of the San Francisco Apple Core, Inc., the Editor, Club Officers or Directors . . . or anyone else in their right minds.

The San Francisco Apple Core Incorporated is a non - profit organization comprised of and supported by owners and users of Apple I I and Apple / / / computers. The Club is run by officers and committees, and the Club endeavors to aid Apple owners through its Program Libraries, Program Disks, meetings, and this publication.

Article Submission: The CIDER PRESS welcomes any and all articles dealing with Apple computers and their associated hardware and software. Articles may be submitted (typed or printed, double - space), or as disk text files using any of the popular word processors or text editors. Any author who has an article published in the CIDER PRESS will receive a free copy of the most current Disk - of - the - Month.

Reprints: All articles appearing in the CIDER PRESS not copyrighted by the author may be reprinted by other *non - profit* Apple User Groups, without prior permission, so long as proper credit is given to both the author and the CIDER PRESS. Permission to reprint a copyrighted article must be obtained in advance, by writing to the authors, c/o the San Francisco Apple Core.

San Francisco Apple Core
1515 Sloat Blvd.
San Francisco, CA 94132

B*A*S*I*C

Copyright (c) 1983 by Joyce Conklin

Since this is the season for gift-giving (at least it still is as I write this) I thought I would mention an item that I have been desiring lately and see if any of you know where I might find it. As my best thoughts for this column seem to come early in the morning as I'm standing in the shower, I would like to have a waterproof cassette tape recorder!

Or, failing that, at least a memory without such a close approximation to a sieve. Until those come through, I will have to continue using the antiquated system (I mean it's really antiquated - actual pieces of paper!!) of sticking notes into a file folder and pulling them out when it's time to compile the column.

. . . . .

There is an ongoing debate in computerdom about the meaning of "computer literacy". We are told that within five years, anyone who doesn't understand computers will belong to the next generation of illiterates. But what is "literate"? Does it mean that you can speak intelligently about computers? that you know there are such things as computers? that you can utilize computers to accomplish a particular task?

Or does it mean, as Prof. Arthur Luehrmann, author of "Hands-On Basic", "Hands-On Pascal", "Hands-On Computer Literacy", and others contends: that you must be able to "speak" the language of computers? And if that last is indeed the case, which language? BASIC? FORTRAN? Forth? COBOL? LOGO? 6502-assembly language? Binary?

While computers have been around in some form since the abacus, and electronic computers since the late 40's, the widespread use of this mind-tool is very recent. VERY recent - how long have you had your computer? Some people worry about the definition of computer literacy because they fear that the ability, or lack thereof, to use computers will be the next dividing line between the haves and the have-nots.

Others are concerned with a definition because it will define the content of curriculum. Do you cover history? What a computer is, physically? How computers are used? BASIC programming? All of the above? [If your answer is "(d) All of the above", a great source at the 6th grade reading level is Spotlight on Computer Literacy by Ellen Richman, published by Random House.]

How long will you be able to have the students? an hour a day? six weeks? nine weeks? one Saturday? once a month? What kind of follow-up can you have? Is the computer available in your classroom for use whenever there is a free space of time or do you share the one computer in the school with fourteen other classes?

Do you think that the area is full of questions with very few answers? You're right! The ideal situation, in my opinion, would be to have the ability to totally integrate the curriculum rather than fragment it - talk about history of computers as a part of a general history discussion; talk about the ethics of computer use (note: computers do not have ethics, nor do they lack ethics; they are only machines, Arthur Clarke notwithstanding) during a discussion of ethics in the workplace (also in the school "place" - see below); discuss the parts of a computer when teaching a technology unit; introduce computer languages when teaching communication.

Unfortunately, the higher/later on the educational ladder you get, the more fragmented the curriculum becomes, rather than less. And the ability to be successful in programming is one of the most divisive fragments of all. Not all people are successful programmers - many do not even like programming (heresy! - but which comes first? don't like, therefore not successful, or the other way around?)

A recent book I came across is The Brain Game, by Rita Aero and Elliot Weiner, published by Quill. It contains 27 intelligence tests "that will reveal your unique abilities". One of these tests is a Computer Science Aptitude Test developed by J. Konvalina, L. Stephens, and S. Wileman at the Univ. of

Nebraska. The introduction to the test states: "Schools offering computer science courses have high enrollments in current classes and report increased demand for course variety.

Schools and corporations that offer training programs for computer personnel have discovered that most of those who are able to complete the course sucessfully have fairly specific qualities to begin with." The test consists of twenty-five questions covering Sequences an Logic, a Calculator Simulator (with reverse-Polish notation yet!), analysis of an algorithm, and word probems.

The interpretation after the test says, "The Computer Science Aptitude Test is intended for just that purpose - the measurement of aptitude in computr science and the probability of sucess in related courses. As it is used in some universities, people who score below average are permitted [ed: notice the segregation involved - "permitted"!] to enroll only in nontechnical 'computer appreciation' courses involving little, if any, computer-language and programming training. . . . If you scored between 0 and 20 you demonstrated a low-to-average level of the basic computer-related skills . . . an above-average to very high score . . . [of] 21 to 25 is associated with high potential for success in technical computer courses."

My caveat would be to take the interpretation with a grain of salt - universities are overwhelmed with students wanting to enroll in the computer science courses and therefore the faculty can be very selective about whom they enroll. But you can learn programming, if that's what you want, by any of several methods; formal courses are a part of only one (not necessarily the best - for all) method.

Reading, talking with others who know something about the field, trying it out (really - the computer won't bite you, although it does have a penchant for testing your frustration level sometimes!) are all ways of learning about the machine which we have and love.

. . .

While attending a meeting of computer teachers, my interest was drawn to a poster which showed the value of pi, carried to several thousand decimal places! I asked the teacher whose room it was if he had a reference which would compute the value of pi, or if he knew how to get it out of the innards of the Apple. I just assumed that it would be there, even though I'd never had an opportunity or need to use pi in a calculation; after all, if it is included in all but the very least expensive hand-held calculators, surely it is in the Apple. Not so! But he gave me a copy of some material from the National Council of Teachers of Mathematics, which I feel is of sufficient interest to share with you this month. (The following paragraphs are taken from this handout with only minor editorial changes.)

The first page offers an activity - always a wonderful thing to have for many reasons, only one of which is that you learn more from doing than from just listening (it's also difficult to keep students still and listening for any prolonged period of time!). Rule a piece of paper horizontally with lines 2-centimeters apart. Cut four 1-centimeter pieces from toothpicks. From a reasonable height (about waist high for a 10-year old) randomly drop the four sticks onto the ruled surface.

A 'hit' occurs when a stick lands on or accross one of the ruled lines. Count the number of hits on each of 25 drops of the four sticks. Divide the total number of sticks dropped (100) by the total number of hits. The quotient should approximate the value of pi. The more total sticks dropped, the better the approximation shoud be. This method is called Buffon's Needle Problem, based on an expansion by Piere Laplace (1749-1827) of an idea of Georges Louis Leclerc, Comte de Buffon (1707-1788).

In 1761 Johnn Heinrich Lambert showed that pi is an irrational number and cannot be written as a repeating decimal. We can correctly compute pi to as many decimal places as we like, but there is no repetitive pattern to its digits. No fractions with integers for numbertor and denominator can exactly equal pi, but there are many fractions that come close enought for practical applications.

In I Kings 7:23 the approximation is given as "And he made a molten sea, ten cubits from one brim to the other: it was round all about ... and a line of thirty cubits did compass it round about." Since pi is defined as the ratio of the circumference of a circle to its diameter, the implied value of pi in this passage is 3. Close enough for government work! (They were working for King Solomon, remember?)

About 240 B.C. Archimedes took a circle and called its diameter one unit. He computed the perimeter of the circumscribed (around the outside of the circle) and inscribed (around the inside

of the circle) polygons of 96 sides and determined that the following relationship existed:

3 10/71 < pi < 3 1/7

In the fifth century A.D. the Chinese astronomer Ch'ung-Chih found that the fraction 355/113 gives an approximation of pi accurate to six decimal places. How do we know how many decimal places are accurate? Here is a calculation of pi that has been carried out to 200 decimal places:

pi is approximately:

3. 14159 26535 89793
23846 26433 83279
50288 41971 69399
37510 58209 74944
59230 78164 06286
20899 86280 34825
34211 70679 82148
08651 32823 06647
09384 46095 50582
23172 53594 08128
48111 74502 84102
70193 85211 05559
64462 29489 54930
38196

My question of the month is: does anyone have a program which will output the above type of approximation of pi? Obviously 355/113 is good enough for most program uses where the value of pi is required for calculations, but to get the long list of digits above, a computer was used, and most computers don't compute and print out past 16 decimal places - Apple's maximum is nine (with the exception noted in the Applesoft Programming Reference Manual, p.33, which came with he Apple II+.)

It would seem to me that either there needs to be a program which will be the equivalent of the DEFDBL command in the TRS version of Microsoft BASIC, or a program which will retain each digit and concatenate it into a string. Anyone have any bright ideas?

In 1674, the German mathematician Gottfried Wilhelm von Leibnitz, one of two independent inventors of the calculus [editorial side-thought: does anyone know why it's called "the" calculus, not just calculus?] used his invention to find an expression for pi as an infinite sum:

pi = 4 (1 - 1/3 + 1/5 - 1/7 + 1/9 - ...)

The simplicity of the Leibnitz series makes it one of the best-known expressions for pi.

. . . . .

Following are two programs relating to the above discussion. The program for Buffon's Needle method is a modified version of a program first published by Ronald J. Carlson and his Plymouth-Canton MI High School computer class (Mathematics Teacher, Nov. 1981, p.639).

Unlike the physical activity of dropping sticks, where the sticks are half as long as the separation between lines, the needle length is the same as the interlinear distance for the graphic display. To get the appropriate value of pi, the multiplier of 2 is inserted in the formula at line 340.

```
100  REM COUNT DE BUFFON'S ESTIMATION OF
     PI
110  INPUT "HOW MANY NEEDLES TO DROP?";N
120  HOME
130  HGR
140  HCOLOR= 2

150  FOR Y = 0 TO 159 STEP 10
160  HPLOT 0,Y TO 279,Y
170  NEXT Y
180 HIT = 0
190  HCOLOR= 3

200  FOR Z = 1 TO N
210 X =  INT ( RND (Z) * 280)
220 Y =  INT ( RND (Z) * 160)
230 ANGLE =  RND (Z) * 1000
240 X1 = X + 10 *  COS (ANGLE)

250 Y1 = Y + 10 *  SIN (ANGLE)
260  IF X1 < 0 OR X1 > 279 THEN 290
270  IF Y1 < 0 OR Y1 > 159 THEN 290
280  HPLOT X,Y TO X1,Y1
290  IF Y = Y1 AND  INT (Y / 10) =
     Y / 10 THEN HIT = HIT + 1

300  IF Y / 10 =  INT (Y / 10) AND
     Y1 / 10 =  INT (Y1 / 10) THEN 320
310   IF   INT  (Y  / 10) <  >  INT (Y1 /
10)
     THEN HIT = HIT + 1
320  NEXT Z
330  VTAB 21
340  PRINT "PI ESTIMATE =";2 * N / HIT
350  END
```

The second program I developed from another idea in the NCTM handout. Lines 500, 510-550, and 600 are in the handout; the remaining lines are the result of my running the program at school and having one of my students enter a ridiculously large number in response to the "How many terms?" question.

Ridiculous = 100 000 000 !!! Because allowing it to run to the end of the loop with 100 000 000 as the top limit would take longer than I wanted to stay at school, I modified the program (with some help from Peter Weiglin) so that

you could save the values at any time and pick up where you left off the next time your computer is pining for something to do.

Although the line numbers are not perfectly incremented by 10's (the result of some final tinkering nd debugging) the program does work as shown; after approximately 7+ hours of computer time, I have run 792 006 terms of the 100 000 000. The current approximatio of pi is 3.141 592 88. I am interested in finding out at what point the significant changes occur beyond the current digit limit of the Apple.

```
70 TEXT : HOME
80 D$ =  CHR$ (13) +  CHR$ (4)
100 TT$ = "APPROXIMATE PI"
104  VTAB 18: PRINT "THIS PROGRAM IS
     DESIGNED TO ALLOW YOU TO";: PRINT
     "USE LEIBNITZ'S METHOD TO
     APPROXIMATE": PRINT "PI"
110  VTAB 7: HTAB 7: FOR L = 1 TO 27:
     PRINT "*";: NEXT : PRINT

115 X = 40 -  LEN (TT$):X = X / 2
130  VTAB 11: HTAB 7: FOR L = 1 TO 27:
     PRINT "*";: NEXT : PRINT
140  FOR L = 8 TO 10: VTAB L: HTAB 7:
     PRINT "*";: HTAB 33: PRINT "*":
     NEXT
150  VTAB 9: HTAB 14: PRINT TT$
160  FOR I = 1 TO 4000: NEXT I
181  HOME : VTAB 2
182  PRINT "SELECT WHETHER THIS IS THE
     FIRST"
185  PRINT "TIME YOU HAVE RUN THIS
     PROGRAM OR IF YOU";: PRINT "ARE
     CONTINUING FROM A PREVIOUS RUN.
190   VTAB  10:  HTAB  10: PRINT "<1> NEW
RUN

192  VTAB 12: HTAB 10: PRINT "<2>
     CONTINUE
194  VTAB 17: HTAB 14: GET AA$
195  IF  VAL (AA$) = 1 THEN  GOSUB
     20000: GOTO 400
196  IF  VAL (AA$) = 2 THEN 300

197  GOTO 194
305  PRINT D$;"NOMON C,I,O"
310  PRINT D$;"OPEN PIFILE"
320  PRINT D$;"READ PIFILE"
330  INPUT N,C,A,B,PI

340  PRINT D$;"CLOSE PIFILE"
400  HOME : VTAB 2: HTAB 3: PRINT
   "LEIBNITZ'S  SERIES  TO  APPROXIMATE
PI"
410  POKE 34,4
420   VTAB  5:  IF  N  =  0  THEN   INPUT
"NUMBER
     OF TERMS =";N: POKE 34,6: GOTO 440
430  VTAB 5: PRINT "NUMBER OF TERMS = ";
     N: POKE 34,6

440  POKE 35,20: VTAB 21: HTAB 7: PRINT
     "PRESS  ANY  KEY TO INTERRUPT": VTAB 6
500  FOR K = C TO N
510  IF K / 100 =  INT (K / 100) THEN
     VTAB 12: HTAB 20: PRINT K
520 PI = PI + B / A
530 B = B * ( - 1)

540 A = A + 2
545  IF  PEEK ( - 16384) > 127 THEN 560
550  NEXT K
560  IF K < N THEN  POKE  - 16368,0:
     GOSUB 18000: GOTO 1000
570  IF K >  = N THEN  POKE 34,3

600 PI = 4 * PI
610   HOME  :  VTAB 10: PRINT "WHEN USING
";
     N;" TERMS, THE VALUE OF PI"
620  PRINT : PRINT "IS APPROXIMATELY "
    ;PI
630  FOR L = 1 TO 2000: NEXT
700  POKE 35,24: CALL  - 958: VTAB 20:
     PRINT "DO YOU WANT TO TRY ANOTHER
     APPROXIMATION";: PRINT "PLEASE
     ANSWER 'Y' OR 'N'

705  GET Q$
710  IF Q$ = "N" THEN  END
720  IF Q$ = "Y" THEN  GOSUB 20000:
     GOTO 400
730  GOTO 700
1000  POKE 34,0: POKE 35,24: HOME

1010  VTAB 4: PRINT "THE INTERIM RESULTS
      ARE:"
1020  PRINT "N = ";N: PRINT "RANGE FOR
      THIS RUN = ";C;" TO ";K
1030  PRINT "A = ";A: PRINT "B =";B:
      PRINT "SERIES SUM SO FAR =";PI
1035  PRINT "PI = ";4 * PI
1040  FOR L = 1 TO 1000: NEXT
1050    PRINT   :  PRINT  "SEE  YOU  NEXT
TIME!": END
1999  END

18000  PRINT D$;"OPEN PIFILE"
18010  PRINT D$;"WRITE PIFILE"
18020  PRINT N
18021  PRINT K
18022  PRINT A
18023  PRINT B
18024  PRINT PI
18030  PRINT D$;"CLOSE PIFILE"
18040  RETURN

20000  REM
20010  PRINT D$;"OPEN PIFILE"
20020  PRINT D$;"DELETE PIFILE"
20040  PRINT D$;"OPEN PIFILE"
20050  PRINT D$;"WRITE PIFILE"
20060 N = 0:C = 2:A = 3:B =  - 1:PI = 1
20070  PRINT N,C,A,B,PI
20080  PRINT D$;"CLOSE PIFILE"
20090  RETURN
```

An interesting quirk appeared when debgging. Notice that in the subroutine to save interim values (18000 -) it seems to be necessary to put each PRINT value in a separate line in order not to have an error result.

However, in the subroutine which initiallizes the file the first time you run the program (these values were given as assignment statements - e.g. 30 PI = 1 - in the original program) all five values may be entered with one print statement (line 20070). Any hot-shots have an answer to that "Why"??

. . . . .

That should keep you busy at the keyboard for a while! I wasn't just kidding when I offered a guest spot to anyone who wanted to write a (or even "the") BASIC column. I am beginning to run out of ideas.

However, our membership information indicates that approximately two-thirds of our members have been i the SFAC for less than one year, so perhaps it is worthwhile re-running some of the earlier columns. I'll try not to duplicate this immediate past year's work. Until Valentine's month, happy hacking.

MARKETING NOTES
by
Don Scellato

I would like to thank everyone who worked at the Club Booth at the Applefest. Everyone who was involved really worked hard. Many gave up the morning of the Saturday prior to the Applefest to attend a briefing session. These sessions really help to make things work more smoothly at the show.

Joe Bianco was back with us during the Applefest and at the last General Meeting. We're all happy that he's well and back at the table.

Three new products will be introduced at the December General Meeting. These products are all on disk for the general use and entertainment of SFAC members. The first product is SFAC VISICALC TEMPLATE DISK NO.2. This disk will include a version of the 1983 Federal Form 1040 to assist members with the preparation of their income taxes. It will also include a Payroll Template and several other useful templates.

The second product is SFAC 1983 DISK NO. 4. This disk contains one of the prize winners from the programming contest. This program is an educational program called EXPLORING THE HUMAN BODY. It also includes a couple of games. These are THE BATTLE OF HOTH and ESPIONAGE ADVENTURE.

The third product is SFAC DISK NO. 5. This disk includes a home management program called GAS BUGGY, a game called BASEBALL ADVENTURE, several utility programs, and a label making program.

Easy Pascal I/O Routines

Copyright 1983 by Michael A. Fischer

[Part Two]

[Note: This article uses ALL CAPITALS to refer to procedures, functions and variables defined in this series of programs and "Quotes" to refer to predefined (intrinsic) procedures, functions and variables.]

In the first article in this series the concept of user friendly, gorilla-proof data entry in Pascal was introduced and the integer input routines were presented. These routines provided screen formatting, error checking, intelligible error messages, and optional range checking and entry prompts.

This article will present and discuss input routines for real (floating point) numbers. Subsequent articles will present long integer input routines and describe how to convert the routines to units so that they can be placed in your System.-Library rather than entered into each source text file that uses them.

Entering the Routines

The REALIODEMO program consists of parts of the INTIODEMO program presented in part one of this series, and the additional text listed in listing 1.

The full text file, including comments, occupies 40 blocks. The Pascal text editor normally holds only 36 blocks; but it can hold up to 40 blocks if swapping is on.

Before starting the editor you should press "S" (swapping) from the command line. The screen should prompt:

Swapping is off.
Toggle swapping?

Answer "Y" (yes) and the editor can hold up to 40 blocks. If you reinitialize the system for any reason, swapping is automatically set off and you must reset it.

Next press "E" (editor) and return for a new work file. Press "C" (copy) and "F" (from file). Give the editor the file name which contains your INTIODEMO file. Include the volume name. You need not add the suffix ".Text" because the editor will do so.

Driving Your Drives

If you have one drive, be sure the disk with the INTIODEMO text file is in the drive before pressing return. If you have more than one drive, be sure the disk with INTIODEMO is in one of the drives before pressing return. If you removed your boot

disk from drive 1 (volume 4), be sure to replace it after copying.

Make the following changes in the copied file:

1. Delete the following global variables (listed at the beginning of the program): INTVALUE, LOWIRANGE and HIGHIRANGE.

2. Add the following global variables, of type "Real": HIGHRRANGE, LOWRRANGE and REALVALUE.

3. For the sake of clarity you should change the program name to something like REALIODEMO, although it will compile and run it you don't change it.

4. Delete the code for the functions IVERT, PIVERT and RIVERT.

5. Change the line in PROCEDURE INTRO reading "Writeln ('string inputs converted to integer.');" by substituting the word reals for the word integers.

6. Globally replace HIGHIRANGE with HIGHRRANGE, LOWIRANGE with LOWRRANGE, and INTVALUE with REALVALUE. To globally change press "J" (jump), "B" (beginning), "/", "R", "/Highirange/Highrrange/" and each occurance of HIGHIRANGE will be replaced with HIGHRRANGE. Repeat for the other variables.

Move the cursor to the beginning of PROCEDURE INTRO, press "I" for Insert, and then enter the text in listing 1.

Quit and save the text file, compile and execute.

Reals Processing Routines

There is one main processing routine, REALCONVERT, which contains 4 subroutines, SPLITIT, ADJUSTEXPONENT, ADJUSTWHOLE, and DECIMALCONVERT. REALCONVERT also calls some of the integer processing routines.

In addition there are 4 functions which are the routines actually called by your main program.

Procedure SPLITIT divides the ENTRY string into two strings -- FIRSTPART and SECONDPART. The character passed in DIVISOR is used as the splitting point and appears in neither part.

SPLITIT is called twice by REALCONVERT. First, after removing blanks it splits the number into a MANTISSA and EXPONENT using the character "E" as the DIVISOR. This provides a means of dealing with numbers entered using scientific notation (i.e., 1.346E-5 instead of 0.00001346). Next it splits MANTISSA into a WHOLE and a DECIMAL part using the decimal point (".") as the DIVISOR character.

Splitit and Pos

The SPLITIT procedure uses the "Pos" function to determine if the desired splitting character (DIVISOR) is present. "Pos" returns an integer (DIVIDEPOINT) which is the starting location of the desired substring in the string, or a 0 if the substring is not present.

If DIVIDEPOINT is not 0, then the substring is present. If the starting location is the frist or last character, then the first or second divided string is set equal to 0. This would occur if a number like .123, 106., E28 or 3.3E was entered. If the substring is not the first or last character, the procedure divides the string using the function "Copy" which copies a number of characters from the string starting at a given location in the string.

REALCONVERT nexts converts EXPONENT into an integer (POWER), keeping its value between -99 and +99 (OKAYLENGTH = 2). Since reals cannot exceed 3.40282E38 (MAXREAL) or be less than 1.17548E-38 (MINREAL), this gives more than enough leeway in entering an exponent value. Later the procedure will check that the number doesn't exceed MAXREAL and MINREAL.

Process the Whole

The procedure next processes the WHOLE part of the number so that its value is between "0" and "9". It does this by transferring all characters but the leftmost one from WHOLE to DECIMAL, increasing POWER by one for each character transferred. This is done by calling Procedure ADJUSTWHOLE.

The ADJUSTWHOLE subroutine first takes the string WHOLE, sets TRANSFER equal to its last character using the "Copy" function. It then transfers that last character to the first character of DECIMAL using the "Concat" function which combines two strings. Finally it deletes the last character from WHOLE using the "Delete" function which deletes a number of characters from a given location in the string.

WHOLE is then converted into a number, REALWHOLE, using INTCONVERT with an OKAYLENGTH of 1.

Process the Decimal

Next the decimal part of the number is processed. If the whole part was 0 (REALWHOLE=0), then any leading zeroes are stripped off the decimal and POWER adjusted lower by one for each leading zero. Since Pascal is limited to 8 digits in accuracy, this ensures that a number entered as .000000000089 is properly converted, rather than evaluated as 0.

The length of WHOLE plus the value of POWER is restricted to between -38 and +39 to keep in the bounds permitted by Pascal. Decimal is then converted into a number, DECIMALREAL, after restricting it to its eight most significant digits. It does this by calling DECIMALCONVERT.

The DECIMALCONVERT subprocedure works much like INTCONVERT (see part one of these articles), except it moves from right to left (necessitating the use of "Downto" instead of "To" in the "For...Do" loop) and divides instead of multiplies by 10 for each digit found.

Combine the Whole

DECIMALREAL and REALWHOLE are combined. If the result is 0, the number is 0 no matter what the exponent is and the conversion returns to the calling routine with VALUE set to 0.

If the result is not 0, VALUE is adjusted to greater than or equal to 1 and less than 10 with POWER adjusted accordingly using ADJUSTEXPONENT. This permits checking that the number does not exceed "Maxreal" or "Minreal". After checking, VALUE is raised to the power of the exponent (POWER), MINUSSIGN is adjusted if necessary, and the procedure ends.

The four main real number functions are just like those in the integer programs except "r" (for real or floating point number) is used as the root, a real number is returned, and high and low range values passed, if any, must be of type "Real".

ErrorMessages

Real input error messages are:

"Misplaced + sign [in mantissa] [in exponent]" -- The + sign used was not the first character in the mantissa or exponent.

"Misplaced - sign [in mantissa] [in exponent]" -- The - sign used was not the first character in the mantissa or exponent.

"Non-number found [in mantissa] [in exponent] [after decimal point]" -- A non-number was found in the mantissa, exponent or after the decimal point.

"Too many digits [in mantissa] [in exponent]" -- Ther mantissa or exponent contained more than the permissible number of digits.

"Value too low; must be ### or more" -- The value entered was less than the minimum specified for the range.

"Value too high; must be ### or less" -- The value entered was greater than the maximum specified for the range.

"Value too large for floating point number" -- The value of the real is greater than the maximum value permitted for reals.

"Value too small for floating point number" -- The value of the real is less than the minimum value permitted for reals.

"No number entered" -- No number remains after removing leading blanks.

Running the Program

The main program, like that in the first of these articles, is a simple demonstration program. Even with this small progrm, however, the size problems caused by all the processing routines is cumbersome, to say the least. And more processing routines can be added.

Later instructions will be given on how to transfer all the routines to your System.-Library, allowing use of the routines without having them clutter and take up room in your program text file.

```
Procedure Realconvert (Entry:String; Var
     Value:Real; Var Errmessage:String;
     Var Okay:Boolean);
(* This routine takes the string entry,
   and, through various steps, converts
   it to a real number, returning an
   appropriate error message if the wrong
   format is used. *)
Const
 Maxreal = 3.40282;
 Minreal = 1.17548;
Var
 Minussign      : Boolean;
 Okaylength,
 Power,
 Realwhole      : Integer;
 Longone        : Long;
 Decimalreal    : Real;
 Decimal,
 Exponent,
 Mantissa,
 Whole          : String;

 Procedure Splitit (Divisor:String; Entry:
      String; Var Firstpart:String; Var
      Secondpart:String);
 (* This routine splits the string into
    two parts based on the first occurence
    of a specified character, passed as
    the parameter Divisor. *)
 Var
  Dividepoint  : Integer;
  Begin
   Dividepoint := Pos (Divisor, Entry);
   If (Dividepoint <> 0)
    Then
     Begin
      If (Dividepoint = 1)
       Then Firstpart := '0'
       Else Firstpart := Copy (Entry,1,
            Dividepoint-1);
      If (Dividepoint = Length (Entry))
       Then Secondpart := '0'
       Else Secondpart := Copy (Entry,
            Dividepoint+1, Length(Entry)
            -Dividepoint)
     End
```

```
      Else
       Begin
        Firstpart := Entry;
        Secondpart := '0'
       End
    End;

  Procedure Adjustexponent (Var Value:Real;
     Var Power:Integer);
(* This procedure adjusts the mantissa so
   that it is between 1 and 10. *)
 Begin
  While (Value < 1) Do
   Begin
    Value := Value * 10;
    Power := Power - 1
   End
 End;

Procedure Adjustwhole (Var Whole:String;
     Var Decimal:String; Var Power:
     Integer);
(* This procedure "moves" the decimal
   point to the left while adjusting the
   exponent. *)
 Var
  Transfer   : String;
 Begin
  Power := Power + 1;
  Transfer := Copy (Whole, Length(Whole),
       1);
  Decimal := Concat (Transfer, Decimal);
  Delete (Whole, Length (Whole), 1)
 End;

Procedure Decimalconvert (Entry:String;
     Var Decreal:Real; Var Errmessage:
     String; Var Okay:Boolean);
(* This procedure converts the decimal
   part of the mantissa string to a
   number. *)
 Var
  Counter   : Integer;
 Begin
  Decreal := 0.0;
  For Counter := Length (Entry) Downto 1
       Do
   If Not (Entry[Counter] in ['0'..'9']
    Then
     Begin
      Errmessage := 'Non-number found
           after decimal point.';
      Okay := False;
      Exit (Decimalconvert)
     End
    Else Decreal := Decreal / 10.0 + Ord
         (Entry[Counter]) - Ord ('0');
   Decreal := Decreal / 10.0;
   Okay := True
  End;

Begin (* Realconvert *)
 Okay := True;
(* Remove blanks *)
  Blankstring (Entry,Okay);
(* If nothing left - error *)
  If Not (Okay)
   Then
    Begin
     Errmessage := 'No number entered.';
     Exit (Realconvert)
    End;
(* Split into exponent and whole and
   decimal parts of mantissa *)
  Splitit ('E',Entry,Mantissa,Exponent);
  Splitit ('.',Mantissa,Whole,Decimal);
(* Convert exponent to number *)
  Okaylength := 2;
  Intconvert (Exponent,Okaylength,Longone,
       Okay,Errmessage);
  Power := Trunc(Longone);
  If Not (Okay)
   Then
    Begin
     Insert (' in exponent',Errmessage,
          Pos ('.',Errmessage));
     Exit (Realconvert)
    End;
(* Remove any sign *)
  Signproc (Whole,Minussign,Okay,
       Errmessage);
  If (Not Okay)
   Then
    Begin
     Insert (' in mantissa',Errmessage,
          Pos ('.',Errmessage));
     Exit (Realconvert)
    End;
(* Remove zeros *)
  Stripzeros (Whole);
(* Make whole part of mantissa one digit
   only *)
  While (Length(Whole) > 1) Do
   Adjustwhole (Whole,Decimal,Power);
(* Convert whole part of mantissa *)
  Okaylength := 1;
  Intconvert (Whole,Okaylength,Longone,
       Okay,Errmessage);
  Realwhole := Trunc (Longone);
  If Not (Okay)
   Then
    Begin
     Insert (' in whole part of mantissa',
          Errmessage,Pos('.',Errmessage));
     Exit (Realconvert)
    End;
(* If whole part of mantissa is 0, then
   strip leading zeros from decimal part
   of mantissa and adjust exponent *)
  If (Realwhole = 0)
   Then
    Begin
     Power := Power - Length (Decimal);
     Stripzeros (Decimal);
     Power := Power + Length (Decimal);
    End;
(* See if number too large *)
  If Length (Whole) + Power > 39
   Then
    Begin
     Okay := False;
     Errmessage := 'Value too large for
          floating point number.';
     Exit (Realconvert)
    End;
(* See if number too small *)
  If Length (Whole) + Power < -38
   Then
    Begin
     Okay := False;
     Errmessage := 'Value too small for
          floating point number.';
     Exit (Realconvert)
    End;
```

```
(* Only eight digits significant *)
  If (Length (Decimal) > 8)
   Then Decimal := Copy (Decimal,1,8);
(* Convert deimal part of mantissa *)
  Decimalconvert (Decimal,Decimalreal,
       Errmessage,Okay);
  If Not (Okay)
   Then Exit (Realconvert);
(* Combine decimal and whole *)
  Value := Realwhole + Decimalreal;
(* If zero, all done *)
  If (Value = 0)
   Then Exit (Realconvert);
(* Adjust mantissa to between 1 and 10 *)
  Adjustexponent (Value,Power);
(* Check that number is not too big -- *)
  If ((Power = 38) and (Abs(Value) >
       Maxreal)) or (Power > 38)
   Then
    Begin
     Okay := False;
     Errmessage := 'Value too large for
          floating point number.'
    End;
(* -- or too small *)
  If ((Power = -38) and (Abs(Value) <
       Minreal)) or (Power < -38)
   Then
    Begin


     Okay := False;
     Errmessage := 'Value too small for
          floating point number.'
    End;
  If Not (Okay)
   Then Exit (Realconvert);
(* Convert to non-scientific format *)
  While (Power > 0) Do
   Begin
    Value := Value * 10.0;
    Power := Power - 1
   End;
  While (Power < 0 ) Do
   Begin
    Value := Value / 10.0
    Power := Power + 1
   End;
(* Replace minus sign *)
  If (Minussign)
   Then Value := - Value
 End;

(* The following routines accept string
   input, and return either an error
   message or the value of the string as a
   real.  Each accepts input or prints a
   prompt at a specified location, and
   prints error messages at another
   specified location (passed as the first
   4 integer parameters).  Function names
   consist of the root "r" (for real), the
   suffix "vert" (for convert), and 0-2
   letters as prefixes.  The prefixes are
   "p" (for prompt - pass prompt string as
   a parameter), "r" (for range - pass low
   and high values of range as parameters)
   and "pr" (for both - pass prompt
   parameter before range parameters). *)
```

```
Function Rvert (X,Y,Errx,Erry:Integer) :
     Real;
 Var
  Okay         : Boolean;
  Temp         : Real;
  Entry,
  Errmessage   : String;
 Begin
  Repeat
   Clearline (X,Y);
   Gotoxy (X,Y);
(* Get entry *)
  Readln (Entry);
(* Convert it *)
  Realconvert(Entry,Temp,Errmessage,Okay);
  If Not (Okay)
   Then
(* Print error message *)
    Begin
     Clearline (Errx,Erry);
     Gotoxy (Errx,Erry);
     Write (Errmessage)
    End
  Until (Okay);
(* Erase error message line *)
  Clearline (Errx, Erry);
  Rvert := Temp
 End;

Function Prvert (X,Y,Errx,Erry:Integer;
     Prompt:String) : Real;
 Var
  Okay         : Boolean;
  Temp         : Real;
  Entry,
  Errmessage   : String;
 Begin
  Repeat
   Clearline (X,Y);
   Gotoxy (X,Y);
(* Prompt for entry *)
  Write (Prompt);
  Readln (Entry);
(* Convert it *)
  Realconvert(Entry,Temp,Errmessage,Okay);
  If Not (Okay)
   Then
(* Print error message *)
    Begin
     Clearline (Errx,Erry);
     Gotoxy (Errx,Erry);
     Write (Errmessage)
    End
  Until (Okay);
(* Erase error message line *)
  Clearline (Errx, Erry);
  Prvert := Temp
 End;

Function Rrvert (X,Y,Errx,Erry:Integer;
     Low,High:Real) : Real;
 Var
  Okay         : Boolean;
  Temp         : Real;
  Entry,
  Errmessage   : String;
 Begin
  Repeat
   Clearline (X,Y);
   Gotoxy (X,Y);
(* Get entry *)
```

```
  Readln (Entry);
(* Convert it *)
  Realconvert(Entry,Temp,Errmessage,Okay);
  If Not (Okay)
   Then
(* Print error message *)
    Begin
     Clearline (Errx,Erry);
     Gotoxy (Errx,Erry);
     Write (Errmessage)
    End;
(* Check range limits *)
  If (Okay) and (Temp < Low)
   Then
    Begin
     Clearline (Errx,Erry);
     Gotoxy (Errx,Erry);
     Write ('Value too low; must be ', Low,
          ' or more.')
    End;
  If (Okay) and (Temp > High)
   Then
    Begin
     Clearline (Errx,Erry);
     Gotoxy (Errx,Erry);
     Write 'Value too high; must be ',High,
          ' or less.')
    End
  Until (Okay) and (Temp <= High) and (Temp
       >= Low);
(* Erase error message line *)
  Clearline (Errx, Erry);
  Rrvert := Temp
 End;

Function Prrvert (X,Y,Errx,Erry:Integer;
     Prompt:String; Low,High:Real) : Real;
 Var
  Okay         : Boolean;
  Temp         : Real;
  Entry,
  Errmessage   : String;
 Begin
  Repeat
   Clearline (X,Y);
   Gotoxy (X,Y);
(* Prompt for entry *)
  Write (Prompt);
  Readln (Entry);
(* Convert it *)
  Realconvert(Entry,Temp,Errmessage,Okay);
  If Not (Okay)
   Then
(* Print error message *)
    Begin
     Clearline (Errx,Erry);
     Gotoxy (Errx,Erry);
     Write (Errmessage)
    End;
(* Check range limits *)
  If (Okay) and (Temp < Low)
   Then
    Begin
     Clearline (Errx,Erry);
     Gotoxy (Errx,Erry);
     Write ('Value too low; must be ', Low,
          ' or more.')
    End;
  If (Okay) and (Temp > High)
   Then
    Begin
     Clearline (Errx,Erry);
     Gotoxy (Errx,Erry);
     Write 'Value too high; must be ',High,
          ' or less.')
    End
  Until (Okay) and (Temp <= High) and (Temp
       >= Low);
(* Erase error message line *)
  Clearline (Errx, Erry);
  Prrvert := Temp
 End;
```


COMPARE!

SPIN-TRONICS = LOW prices + LOW overhead and GREAT VALUES

**** BLANK DISKS ****			***** PRINTERS *****	
Nashua	SS/SD	19.50	Seikosha	$225.
			Gemini 10X	325.
Ultra	SS/DD	22.95	Prowriter I	435.
Ultra	DS/DD	29.95	Prowriter II	699.
			Epson RX-80	425.
Elephant	SS/SD	22.95	Epson RX-80F/T	499.
Elephant	SS/DD	25.95	Epson FX-80	650.
Elephant	DS/DD	34.95	Okidata 92	535.
			Mannesman Tally	399.
Verbatim	SS/DD	24.95	Brother DX-15	545.
Verbatim	DS/DD	38.95	TTX	589.
Head Clean. Kit		11.95	Daisywriter	1295.
			NEC 3510	1515.
Dysan	SS/SD	27.95	Starwriter	1299.
Dysan	SS/DD	39.95		
Dysan	DS/DD	49.95	****** MODEMS ******	
***** EXTRA'S *****			RS-232	99.
Disk Storage Box		14.95	J-Cat	129.
Box Grn Bar Paper		17.95	Hayes Smrtmdm.	235.
Disk Carring Case		3.49		

CASH PRICES ONLY - ADD 3% FOR MASTERCARD AND VISA

SPIN-TRONICS

2490 Channing Way Suite 409 at Telegraph, Berkeley
(415) 843-2746 • MODEM LINE (415) 843-2749

FORTH WORLD

-Ed Bronstein.

GOING FORTH

The ensuing discussion(s) will be using as a model the Apple Core Forth tutorial diskette called "Blossoming Forth." It is available from the Apple Core or from the Core Forth interest group.

Forth commands are created by using previous commands. These commands, called "words" can be executed either from the keyboard or from a program. A listing of the words in the system may be looked at by typing the word "VLIST" (do not enter the quotation marks) RETURN. The listing may be stopped from scrolling or started again by pressing any key. CTRL C will exit altogether.

Numbers are the things that make computers work: utimately, all computer instructions end up being numbers. The expert programmer, in any language, must have a good grasp of number handling.

Forth handles numbers differently from other languages such as BASIC or Pascal in that Forth keeps them on a stack, one on top of the other like a deck of cards: the card on top is the first one dealt (the dealer is honest, of course). Numbers are placed on top of the stack and removed from the top of the stack, last in, first out (LIFO).

The slots that make up a Forth stack can contain any number between 32767 and -32768. Numbers larger or smaller than these are handled differently -- something we may get into later.

A period (.), called "dot", is a Forth word that prints out the top number from the stack. If, say, the numbers 7 20 30 are entered, three dots separated by spaces will print out 30 20 7.

Each dot removes a number from the stack and prints it. If there is no number to be removed, the statement EMPTY will appear on the screen.

Each time a dot prints a number, the number is removed from the stack forever. A word that will print out the contents of the stack without destroying them in the process is .s (dot ess).

There will be times that the stack will be full of numbers you'll want to get rid of. Rather than print out a lot of dots, use the command SP! (pronounced ess pee store). This will clear the stack. Also, a character or set of characters that are typed in but not found in the dictionary will cause the computer to complain with a beep and clear the stack.

Numerical operators in Forth are the usual + - * / (add, subtract, multiply, divide); however, they are placed after the numbers they work on rather than in between. In other words, instead of the expression 7 + 5 Forth will use the notation 7 5 +. This is called "postfix notation" or "reverse Polish" notation after the Polish mathematician who originated the method. Postfix notation has become well known through its use on Hewlitt-Packard calculators.

Try adding, subtracting, multiplying and dividing some numbers. The numbers and operators must be separated by spaces. Use a dot when you wish to print out the result -- which should be sitting on top of the stack.

```
7 5 + . 12

7 5 - . 2

7 5 * . 35

18 6 / . 3
```

And one other: /MOD will divide and leave any remainder on the stack. Note that two dots are necessary to print out both results.

```
7 5 /MOD . . 1 2
```

Many times it is necessary to change the position of stack numbers. Here are the words by which you can manipulate stack numbers:

DUP	Duplicates number on top of stack. Leaves both numbers
SWAP	Exchanges top two numbers
OVER	Copies 2nd number and puts in on top
ROT	Moves third number to top
N PICK	Copies Nth number and puts it on top (2 PICK is same as OVER)
N ROLL	Moves Nth number to top (3 ROLL is the same as ROT)
DROP	Erases top number
MAX	Leaves larger of two numbers
MIN	Leaves lesser of two numbers

You may want to play with these words to get a feel for what they do. Next month we'll get into how they are used in programming.

Filemanager Machine Language Interface
Part 1 of a Series

by Bryan Costales
(c) B. Costales 1983

INTRODUCTION

Welcome and beware. You are about to enter the hazardous world of interfacing assembly language to the DOS Filemanager. WARNING: These articles are intended for the intermediate to advanced assembly language programmer. I accept no responsibility for the garbaged disks which will inevitably litter the path of the novice.

In this series of articles I will present a grouping of routines which provide an alternate approach to interfacing assembly language programs to the DOS Filemanager. As an acronym, I will refer to this grouping as FMLI, for "Filemanager Machine Language Interface." The aim is to create a body of subroutines which enable the 6502 assembly language programmer to handle file input/output with the ease and facility of higher level languages. Patterned after the calling conventions of the C language, FMLI could, for instance, be used to develop your own higher level language compiler. Or, perhaps, a better operating system.

In the C language (at it's lowest level), a file is opened with

```
fd = open(filename,mode);
```

where mode is "read" or "write" or "read/write". And where fd is a number (file descriptor) which describes the opened file. Thereafter, all manipulations of the file are handled using only that number.

For example, if you wished to read one character from that file, then close the file, you would

```
character = fgetc(fd);
close(fd);
```

(Yes, purists, I have omitted the identifier declarations.)

Things are certainly not near that simple in assembly language. If you ever try to open a file using the DOS Filemanager, you will find yourself writing over 30 lines of code. And that's merely to open the file.

With FMLI installed, opening a file by name requires a mere 5 lines of code:

```
BRK                 ;call FMLI.
BYT FOPEN           ;command character.
ADR FILENAME        ;address of string.
ADR MODEBYTE        ;address mode byte.
STA FD              ;save file descript.
;
```

If some of this looks unusual, it may be because I am using the Lisa 2.6 assembler. As we go along, you owners of other assemblers should check the actual bytes produced in the assembler listings.

Another unusual feature is the use of the BRK statement to call FMLI. I have done this so that any program written which uses FMLI will operate regardless of where FMLI may be in memory. In some applications, FMLI could even overlay the front end of DOS. In others, it may reside on the language card, or on a PROM in some slot. This is not to say that FMLI is relocatable, only that it may be assembled for almost any location, and still be accessible by programs written for it.

FMLI does not adhere to the page 3 conventions. It expects DOS 3.3 to be in a known and fixed place in memory. It accesses the Filemanager directly for maximum I/O speed. Commercial programs which access DOS using the page 3 vectors, such as FID will still work with FMLI installed. Programs which utilize DOS's front end, like Pie Writer, will only work if FMLI does not overwrite that front end or the DOS buffers. For best results, expect to use FMLI with a clean, unmodified, 48k version of DOS. As of this writing, it is un-tested with Turbodos, David Dos, Anix, The Orca OS, and the Aztec Shell, etc.

In this, the first installment, I shall outline the system calls, and begin the assembly language source listings.

SYSTEM CALLS

fmain: The coldstart routine. Resets the break vector, and i/o hooks. Initializes the buffers.

fsetup: Returns the current himem address.

fopen: Attempts to open a file. Returns an error if file is not found.

fcreate: Opens a file. If file not found, creates it assigning it a specified file type.

fgetc: Reads and returns a single character from the indicated file.

fgetpos: Returns the current position of the file pointer into the file.

fputc: Writes a single character to the indicated file.

fread: Reads a range (or block) of bytes into a declared buffer.

fwrite: Writes a range (or block) of bytes to the indicated file, from a declared buffer.

fseek: Sets the file pointer to a specified position in the file. May be set relative to the beginning or the current position.

frewind: Sets the file pointer to the beginning of the file.

fclose: Closes the named file. Close is the only way to flush a buffer to disk.

fclose2: Closes all open files.

ftype: Returns a file's type, including the lock bit.

fdata: Returns the address of a file's data buffer.

fts: Returns the address of a file's track/sector list, and the current offset into that list.

fchmod: lock/unlock a file.

fsetsdv: Set a default slot, drive, and volume. Use before a catalog, and to change the current settings prior to an open.

fcat: Catalogs the disk. Lists names only, 4 to a line, truncates at 19th char.

fcatl: The usual catalog listing.

funlink: Deletes a file.

fgetsdv: Returns the slot, drive, and volume of a file.

fgetnam: Returns the address of the filename for the file.

frename: Renames file from oldname to newname.

ferror: Expects an error number in the a-register. Calls which return an error, return the carry set, and the appropriate error number in the a-register. May not return, see below.

fquit: Resets the vectors. Resets the i/o hooks. Reconnects DOS. Causes DOS to rebuild its buffers. Exits with a cold start to the mother board rom language.

RETURN CODES AND ERRORS

The call to ferror prints an error message with consequencee determined by the mode byte. If the mode byte has the hi bit set, text will be printed to the standard output. That is, to wherever the output hooks at $36,$37 point. If the mode byte has the hi bit clear, text will be printed to the video screen. If bit 6 is set, the message will be printed and the program will abort thru fquit. A call to ferror with the a-register set by an error in a previous call will print to the video screen and return.

Three file descriptors are built-in and reserved. A file descriptor of 00 is "stdout" or the standard output as set by $36,$37. If a printer is enabled, for example, an fputc to a file descriptor of 00 will send a character to the printer. A file descriptor of 01 is "stderr" or the standard error output device. This is always the current active video screen. A file descriptor of 02 is "stdin" or the standard input as set by $37,$38. A call to fgetc with a file descriptor of $02 will fetch a character from the current active input device. Generally this would be the keyboard, but could as easily be a modem, or a remote terminal.

The built in error messages take the form:

"filename" {s#,d#,v#} text_of_error

An example for a call to fopen for "junk,d2" if "junk" doesn't exist on that drive, is:

"junk" d2 can't open

Since FMLI will search all active drives for a specified name if no slot, drive, or volume is specified, a call to fopen for "junk" would print the error:

"junk" can't open

List of error messages:

```
can't open
can't close
file protected
disk full
catalog full
wrong filetype
improper filename
impossible place
dreaded i/o error
system failure
can't allocate buffer
write only
read only
bad system call
out of range
```

LISA 2.6 ASSEMBLY SOURCE LISTING FOR FMLI -- part 1

```
0800                 1  ;
0800                 2          LST
0800                 3  ;
0800                 4          TTL "SOURCE.FMLI.1"
0800                 5  ;
0800                 6  ;
0800                 7  ******************************************************
0800                 8  *                                                    *
0800                 9  * Filemanager Machine Language Interface 3.3/1.0     *
0800                10  *                                                    *
0800                11  * PART 1:                                           *
0800                12  *                  The Equates                       *
0800                13  *                  BFMAIN coldstart initializer      *
0800                14  *                  supporting subroutines            *
0800                15  *                                                    *
0800                16  * assembler: LISA 2.6 (c) Lazerware                  *
0800                17  * FMLI3.3 version 1.0 (c) Bryan Costales 1983        *
0800                18  *                                                    *
0800                19  * Placed into the public domain for non-            *
0800                20  * commercial uses only, November 16, 1983.          *
0800                21  * Requests for use in commercial products should     *
0800                22  * be directed to the author c/o the Cider Press.     *
0800                23  *                                                    *
0800                24  ******************************************************
0800                25  ;
0800                26  *
0800                27  * FMLI zero page equates
0800                28  *
0800                29  ;
0800                30  ; The following are NOT safe. They do not conflict
0800                31  ; with DOS, but may conflict with basic, and some
0800                32  ; commercial programs.
0800                33  ; They are not cast in stone. Move them if necessary.
0800                34  ; Those which must be grouped and in a special order,
0800                35  ; are indicated.
0800                36  ;
0800                37  ; The following must be together and in this order.
0800                38  ;
0000                39  FCURVOL   EPZ $00              ;current file's volume.
0001                40  FCURDRIV  EPZ FCURVOL+$01      ;current file's drive.
0002                41  FCURSLOT  EPZ FCURDRIV+$01     ;current file's slot.
0003                42  FCURTYPE  EPZ FCURSLOT+$01     ;current file's type.
0004                43  FCURNP    EPZ FCURTYPE+$01     ;address of file name.
0006                44  FCURFD    EPZ FCURNP+$02       ;current file descriptor.
0007                45  FCURROFF  EPZ FCURFD+$01       ;current read offset.
0005                46  FCURWK    EPZ FCURNP+$01       ;address of work buffer.
0007                47  FCURTS    EPZ FCURWK+$02       ;address of t/s buffer.
0009                48  FCURDAT   EPZ FCURTS+$02       ;address of data buffer
0800                49  ;
```

```
0800              50  ; These may be in any order, but must still
0800              51  ; be zero page variables.
0800              52  ;
000B              53  FSP       EPZ FCURDAT+$02     ;pointer to file's status

000D              54  FRWSTAT   EPZ FSP+$02         ;i/o status for file.
000E              55  CONSOUT   EPZ FRWSTAT+$01     ;stderr hook
0010              56  BP        EPZ CONSOUT+$02     ;brk processing pointer
0012              57  OLDIRQ    EPZ BP+$02          ;old irq save vector
0014              58  ASAVE     EPZ OLDIRQ+$02      ;register save areas
0015              59  XSAVE     EPZ ASAVE+$01
0016              60  YSAVE     EPZ XSAVE+$01
0017              61  SSAVE     EPZ YSAVE+$01
0018              62  PSAVE     EPZ SSAVE+$01
0019              63  AP        EPZ PSAVE+$01       ;general address pointer
001B              64  VP        EPZ AP+$02          ;general variable pointer
001D              65  HIMEM     EPZ VP+$02          ;dynamicaly set by open.
0800              66  ;
0800              67  *
0800              68  * monitor and dos zero page equates
0800              69  *
0800              70  ;
0033              71  PROMPT    EPZ $33             ;prompt character.
0036              72  CSWL      EPZ $36             ;stdout hook.
0039              73  KSWL      EPZ $39             ;stdin hook.
0800              74            ENZ                 ;end of zeropage equates.
0800              75  ;
0800              76  *
0800              77  * other equates
0800              78  *
03F0              79  BRKVECT   EQU $3F0            ;autostart brk vector.
03FE              80  DBLVECT   EQU $3FE            ;doubletime brk vector.
0800              81  ;
AAFD              82  CALLFM    EQU $AAFD           ;call filemanager.
B5BB              83  FMOPCODE  EQU $B5BB           ;These addresses for
B5BC              84  FMSUBCOD  EQU $B5BC           ; the filemanager
B5BD              85  FMRECORD  EQU $B5BD           ; parameter list are
B5BF              86  FMOFFSET  EQU $B5BF           ; valid for DOS 3.3
B5BF              87  FMVOLUME  EQU $B5BF           ; only.
B5C0              88  FMDRIVE   EQU $B5C0
B5C1              89  FMRANGEL  EQU $B5C1
B5C1              90  FMSLOT    EQU $B5C1
B5C2              91  FMTYPE    EQU $B5C2
B5C3              92  FMNAME    EQU $B5C3
B5C3              93  FMBYTE    EQU $B5C3
B5C3              94  FMRANGEA  EQU $B5C3
B5C5              95  FMRETURN  EQU $B5C5
B5C7              96  FMWORK    EQU $B5C7
B5C9              97  FMTSLIST  EQU $B5C9
B5CB              98  FMDATA    EQU $B5CB
0800              99  ;
FDED             100  COUT      EQU $FDED
FE95             101  OUTPORT   EQU $FE95
FE8B             102  INPORT    EQU $FE8B
0800             103  ;
0800             104  ; TOPMEM determines the address from which the
0800             105  ; buffers will build down. Since FMLI is romable, you
0800             106  ; must set this value here. The actual address
0800             107  ; for the HEAP is in the sixth byte of actual code,
0800             108  ; so may be reconfigured by any loading routine,
0800             109  ; provided FMLI is loaded into ram.
0800             110  ;
9000             111  TOPMEM    EQU $9000           ;developmental version.
8FE7             112  HEAP      EQU TOPMEM-!25
0800             113  *
```

```
0800              114  * calling constants
0800              115  *
0080              116  FSETUP    EQU $80          ;returns current himem.
0081              117  FOPEN     EQU $81          ;open, error if not found

0082              118  FCREATE   EQU $82          ;open, create if not fnd.
0083              119  FGETC     EQU $83          ;fetch next character.
0084              120  FGETPOS   EQU $84          ;returns current position

0085              121  FPUTC     EQU $85          ;put character into file.
0086              122  FREAD     EQU $86          ;read a block of bytes.
0087              123  FWRITE    EQU $87          ;write a block of bytes.
0088              124  FSEEK     EQU $88          ;position file pointer.
0089              125  FREWIND   EQU $89          ;pos. to start of file.
008A              126  FCLOSE    EQU $8A          ;close named file.
008B              127  FCLOSE2   EQU $8B          ;close all open files.
008C              128  FTYPE     EQU $8C          ;returns type of file.
008D              129  FDATA     EQU $8D          ;returns address databuff

008E              130  FTS       EQU $8E          ;returns address t/s list

008F              131  FCHMOD    EQU $8F          ;lock/unlock file.
0090              132  FSETSDV   EQU $90          ;set slot,drive,volume.
0091              133  FCAT      EQU $91          ;catalog file names only.
0092              134  FCATL     EQU $92          ;full catalog
0093              135  FUNLINK   EQU $93          ;unlink (delete) a file.
0094              136  FGETSDV   EQU $94          ;returns slot,drive,vol.
0095              137  FGETNAM   EQU $95          ;returns address filename

0096              138  FRENAME   EQU $96          ;rename oldname, newname.
0097              139  FERROR    EQU $97          ;print text of error.
0098              140  FEOF      EQU $98          ;returns true if EOF.
0099              141  FQUIT     EQU $99          ;quits FMLI, restores DOS

0800              142  ;
0800              143  ; error constants
0800              144  ;
0001              145  EOPEN     EQU $01          ;can't open
0002              146  ECLOSE    EQU $02          ;can't close
0003              147  EPROTECT  EQU $03          ;file protected.
0004              148  EFULL     EQU $04          ;disk full.
0005              149  ECATFULL  EQU $05          ;catalog full.
0006              150  ETYPE     EQU $06          ;wrong file type.
0007              151  ENAME     EQU $07          ;bad file name.
0008              152  EPLACE    EQU $08          ;impossible place.
0009              153  EIOERR    EQU $09          ;dreaded i/o error.
000A              154  EFAILURE  EQU $0A          ;system failure.
000B              155  EBUFFERS  EQU $0B          ;can't allocate buffer.
000C              156  EWRITE    EQU $0C          ;write only.
000D              157  EREAD     EQU $0D          ;read only.
000E              158  EBADCALL  EQU $0E          ;bad system call.
000F              159  ERANGE    EQU $0F          ;out of range.
0800              160  ;
0800              161  ; masks and constants
0800              162  ;
0010              163  BRKMASK   EQU $10          ;test B flag
00FE              164  CLCMASK   EQU $FE          ;clear C flag
0001              165  SECMASK   EQU $01          ;set C flag
0000              166  EOF       EQU $00          ;end of file
008D              167  CR        EQU $8D          ;carriage return
0000              168  EOS       EQU $00          ;end of string
0256              169  BUFFSIZE  EQU $100+$100+!45+!30+$0B
0800              170  ;
0800              171  ; This developmental version is origined to
0800              172  ;  run at $9000 with the HEAP and buffers
0800              173  ;  building down from there.
0800              174  ;
```

```
9000                 175           ORG TOPMEM
9000                 176           OBJ $800
9000                 177  ;
9000                 178  ;---------------------------------------
9000                 179  BFMAIN:
9000                 180  ; Disconnects dos from i/o hooks, sets
9000                 181  ; consout for standard error output.
9000                 182  ; Sets the brk and irq vectors to point
9000                 183  ; to bfvector. Saves the old irq vector
9000                 184  ; in oldirq. This unusual approach is to
9000                 185  ; support the Doubletime Printer(tm) rom.
9000                 186  ; Builds the buffer address list at HEAP
9000                 187  ; and marks all buffers as available.
9000                 188  ;---------------------------------------
9000 4C 07 90        189           JMP BFMAIN1          ;skip defaults table.
9003                 190  ;
9003 03              191  VIDOUT   HEX 03              ;80 col in slot 3
9004 03              192  VIDIN    HEX 03              ; use 00 for 40 col.
9005 E7 8F           193  HEAPADR  ADR HEAP            ;address of the HEAP.
9007                 194  ;
9007 AD 03 90        195  BFMAIN1  LDA VIDOUT          ;disconnect dos by
900A 20 95 FE        196           JSR OUTPORT         ; directly setting hooks.
900D A9 8D           197           LDA #$8D
900F 20 ED FD        198           JSR COUT
9012 AD 04 90        199           LDA VIDIN
9015 D0 03           200           BNE >1
9017 20 8B FE        201           JSR INPORT          ;save vidout as standard
901A A5 36           202  ^1       LDA CSWL            ; error output.
901C 85 0E           203           STA CONSOUT
901E A5 37           204           LDA CSWL+$01
9020 85 0F           205           STA CONSOUT+$01
9022 AD FE 03        206           LDA DBLVECT         ;save old irq
9025 85 12           207           STA OLDIRQ
9027 AD FF 03        208           LDA DBLVECT+$01
902A 85 13           209           STA OLDIRQ+$01
902C A9 6D           210           LDA #BFVECTOR       ;set brk & irq
902E 8D F0 03        211           STA BRKVECT         ; to point to bfvector.
9031 8D FE 03        212           STA DBLVECT
9034 A9 90           213           LDA /BFVECTOR
9036 8D F1 03        214           STA BRKVECT+1
9039 8D FF 03        215           STA DBLVECT+1
903C                 216  ;
903C                 217  ; The heap is organized as:
903C                 218  ;     top-> address buffer8
903C                 219  ;           allocation byte
903C                 220  ;           address buffer7
903C                 221  ;           allocation byte
903C                 222  ;           ...
903C                 223  ;           address buffer1
903C                 224  ;           allocation byte
903C                 225  ;  bottom-> number active buffers byte
903C                 226  ;
903C                 227  ; where:
903C                 228  ;     allocation byte=00 buffer not allocated.
903C                 229  ;                    =01 buffer allocated
903C                 230  ;                    =all else, system clobbered.
903C                 231  ;
903C AD 05 90        232           LDA HEAPADR         ;create the heap at
903F 85 1B           233           STA VP              ; HEAPADR.
9041 85 19           234           STA AP
9043 AD 06 90        235           LDA HEAPADR+$01
9046 85 1C           236           STA VP+$01
9048 85 1A           237           STA AP+$01
904A A0 00           238           LDY #$00
904C 98              239           TYA
904D 91 1B           240           STA (VP),Y
904F C8              241           INY
9050 A9 00           242  ^2       LDA #$00
9052 91 1B           243           STA (VP),Y
9054 C8              244           INY
9055 A5 19           245           LDA AP
9057 38              246           SEC
```

```
9058 E9 56      247           SBC #BUFFSIZE
905A 85 19      248           STA AP
905C 91 1B      249           STA (VP),Y
905E A5 1A      250           LDA AP+$01
9060 E9 02      251           SBC /BUFFSIZE
9062 C8         252           INY
9063 85 1A      253           STA AP+$01
9065 91 1B      254           STA (VP),Y
9067 C8         255           INY
9068 C0 1A      256           CPY #!26
906A 90 E4      257           BLT <2
906C 60         258           RTS                    ;exit with a return.
906D            259  ;
906D            260  ;---------------------------------------
906D            261  BFVECTOR:
906D            262  ; Arrive here after a BRK. With the
906D            263  ; Doubletime Printer(tm) ROM, also arrive
906D            264  ; here with a hardware interrupt request.
906D            265  ; Exit to OLDIRQ if not a BRK.
906D            266  ; Saves the registers and jumps with a rts
906D            267  ; to the requested routine.
906D            268  ;---------------------------------------
906D 78         269           SEI                    ;dissable interrupts
906E D8         270           CLD                    ;clear decimal mode
906F 85 14      271           STA ASAVE              ;save the a-reg.
9071 68         272           PLA                    ;get saved p-reg.
9072 85 18      273           STA PSAVE              ; and save it.
9074 29 10      274           AND #BRKMASK           ;test for BRK
9076 D0 08      275           BNE >1
9078 A5 18      276           LDA PSAVE              ;not a BRK, goto
907A 48         277           PHA                    ; saved irq vector.
907B A5 14      278           LDA ASAVE
907D 6C 12 00   279           JMP (OLDIRQ)
9080 86 15      280  ^1       STX XSAVE              ;yes a BRK, save
9082 84 16      281           STY YSAVE              ; x,y regs.
9084 68         282           PLA                    ;fetch calling address
9085 85 10      283           STA BP                 ; and save in BP
9087 68         284           PLA
9088 85 11      285           STA BP+$01
908A BA         286           TSX                    ;save stack pointer.
908B 86 17      287           STX SSAVE
908D 20 17 91   288           JSR DECBP              ;BP-1 points to command.
9090 A0 00      289           LDY #$00
9092 B1 10      290           LDA (BP),Y             ;fetch command
9094 30 10      291           BMI >4                 ;hi-bit on, pos command.
9096 BA         292  ^3       TSX                    ;not a command:
9097 E8         293           INX                    ; restore stack.
9098 E8         294           INX
9099 9A         295           TXS
909A A5 18      296           LDA PSAVE
909C 48         297           PHA
909D A5 14      298           LDA ASAVE              ; restore registers.
909F A6 15      299           LDX XSAVE
90A1 A4 16      300           LDY YSAVE
90A3 4C 4C FA   301           JMP $FA4C              ; exit normal brk handler

90A6 29 7F      302  ^4       AND #$7F               ;clear hi bit.
90A8 C9 1A      303           CMP #FQUIT-$7F         ;test command range.
90AA B0 EA      304           BGE <3                 ;reject if out of range.
90AC 0A         305           ASL                    ;fetch address of routine
90AD AA         306           TAX                    ; push onto stack
90AE BD B7 90   307           LDA BFJTABLE,X         ; and jump there with rts

90B1 48         308           PHA
90B2 BD B8 90   309           LDA BFJTABLE+$01,X
90B5 48         310           PHA
90B6 60         311           RTS
90B7            312  ;
90B7            313  BFJTABLE:
90B7 FF FF      314           ADR BFSETUP-$01
90B9 FF FF      315           ADR BFOPEN-$01
90BB FF FF      316           ADR BFCREATE-$01
```

```
90BD FF FF      317           ADR BFGETC-$01
90BF FF FF      318           ADR BFGETPOS-$01
90C1 FF FF      319           ADR BFPUTC-$01
90C3 FF FF      320           ADR BFREAD-$01
90C5 FF FF      321           ADR BFWRITE-$01
90C7 FF FF      322           ADR BFSEEK-$01
90C9 FF FF      323           ADR BFREWIND-$01
90CB FF FF      324           ADR BFCLOSE-$01
90CD FF FF      325           ADR BFCLOSE2-$01
90CF FF FF      326           ADR BFTYPE-$01
90D1 FF FF      327           ADR BFDATA-$01
90D3 FF FF      328           ADR BFTS-$01
90D5 FF FF      329           ADR BFCHMOD-$01
90D7 FF FF      330           ADR BFSETSDV-$01
90D9 FF FF      331           ADR BFCAT-$01
90DB FF FF      332           ADR BFCATL-$01
90DD FF FF      333           ADR BFUNLINK-$01
90DF FF FF      334           ADR BFGETSDV-$01
90E1 FF FF      335           ADR BFGETNAM-$01
90E3 FF FF      336           ADR BFRENAME-$01
90E5 FF FF      337           ADR BFERROR-$01
90E7 FF FF      338           ADR BFEOF-$01
90E9 FF FF      339           ADR BFQUIT-$01
90EB            340   ;
90EB            341   ;-----------------------------------------------
90EB            342   BFEXIT:
90EB            343   ; All routines exit through here.
90EB            344   ; Any registers which are to return
90EB            345   ; information have been conditioned
90EB            346   ; in the save areas.
90EB            347   ;-----------------------------------------------
90EB A6 17      348           LDX SSAVE
90ED 9A         349           TXS
90EE A5 18      350           LDA PSAVE
90F0 48         351           PHA
90F1 A5 14      352           LDA ASAVE
90F3 A6 15      353           LDX XSAVE
90F5 A4 16      354           LDY YSAVE
90F7 20 10 91   355           JSR INCBP
90FA 28         356           PLP
90FB 6C 10 00   357           JMP (BP)
90FE            358   ;
90FE            359   ;-----------------------------------------------
90FE            360   BFCLC:
90FE            361   ; Clears the carry flag in the p reg
90FE            362   ; which has been saved in PSAVE.
90FE            363   ;-----------------------------------------------
90FE 48         364           PHA
90FF A5 18      365           LDA PSAVE
9101 29 FE      366           AND #CLCMASK
9103 85 18      367           STA PSAVE
9105 68         368           PLA
9106 60         369           RTS
9107            370   ;
9107            371   ;-----------------------------------------------
9107            372   BFSEC:
9107            373   ; Sets the carry flag in the p reg
9107            374   ; which has been saved in psave.
9107            375   ;-----------------------------------------------
9107 48         376           PHA
9108 A5 18      377           LDA PSAVE
910A 09 01      378           ORA #SECMASK
910C 85 18      379           STA PSAVE
910E 68         380           PLA
910F 60         381           RTS
9110            382   ;
9110            383   ;-----------------------------------------------
9110            384   INCBP:
9110            385   ; increments the pointer to passed data
9110            386   ; by one.
9110            387   ;-----------------------------------------------
```

```
9110 E6 10       388            INC BP
9112 D0 02       389            BNE >1
9114 E6 11       390            INC BP+$01
9116 60          391  ^1        RTS
9117             392  ;
9117             393  ;---------------------------------------
9117             394  DECBP:
9117             395  ; decrement the pointer to passed data
9117             396  ; by one.
9117             397  ;---------------------------------------
9117 48          398            PHA
9118 A5 10       399            LDA BP
911A D0 02       400            BNE >1
911C C6 11       401            DEC BP+1
911E C6 10       402  ^1        DEC BP
9120 68          403            PLA
9121 60          404            RTS
9122             405  ;
9122             406  ;---------------------------------------
9122             407  PULLADR:
9122             408  ; Increment the passed parameter pointer
9122             409  ; (BP) and fetches the next address.
9122             410  ; Places that address into AP.
9122             411  ;---------------------------------------
9122 20 10 91    412            JSR INCBP
9125 A0 00       413            LDY #$00
9127 B1 10       414            LDA (BP),Y
9129 85 19       415            STA AP
912B 20 10 91    416            JSR INCBP
912E B1 10       417            LDA (BP),Y
9130 85 1A       418            STA AP+1
9132 60          419            RTS
9133             420  ;
9133             421  ; to be continued next issue...
```

January Surprises

January's meeting is a mystery. If Apple's new releases (rumored to be Pro-Dos and MacIntosh) are public by then, we will be presenting the first inside look at them. In addition, Legisoft will be demonstrating Will Writer, which uses the Apple to write a will based on your answers to particular questions. They will be selling the program at a special member price.

Otherwise, just think of some of those special programs you saw at Applefest -- perhaps that great new graphics item, word processor, or home spectacular program. There will be some special demonstration. Whatever happens, we'll try to entertain, educate and even excite you.

See you Saturday, January 21, 1984 (hello George Orwell) at the usual time and place.

REVIEW:

Think Tank

Copyright 1983 by Michael Fischer

Most database systems are fairly inflexible when it comes to processing data. They work very well as long as the data fits into the specified record format, and the infomration being processed is all of a similar type.

But what do you do when you're dealing with a variety of ideas, related but varying in format and content. The old answer -- use index cards. The new answer -- use an idea processor.

Actually there is only one idea processor, but it is so well executed there is only need for this one. It is a program called *ThinkTank*, subtitled "The First Idea Processor."

The program is an electronic outline, using most of the video screen for a data input area. ThinkTank's utility comes from the powerful commands which manipulate the outline.

The program is designed to run on an Apple][,][+, //e or ///. Two drives or a hard disk are required. The Videx Videoterm and the //e 80 column card are supported. A minimum of 64K RAM (96K on the Apple ///) is required.

Menu Driven

The program is menu driven with the ability to bypass many of the menus with command keys once familiarity with the commands is gained. The various outline manipulation commands include:

Collapse - hide material under an outline headline. The material under a headline can be either sub-headings or textual paragraphs. Either the subheadings, the paragraphs, or both may be collapsed. When parts of a large outline are collapsed, the overall structure of the outline becomes more discernable.

Expand - reveal material under an outline headline which has previously been collapsed. The use of collapse and expand allows visualizing the outline and its details at will.

New - add either an outline headline or a text paragraph to the outline. Headlines can be added either superior, equal or subordinate to other outline headlines.

Move - rearrange the outline by moving headlines up or down and rearrange the relationship of the various headlines.

Delete - remove either an outline headline or a paragraph.

Edit - modify the text of outline headlines or paragraphs. A number of subcommands are available when in edit mode. These include many standard text editing features such as insert, delete, typeover, find and exchange.

Keyword - search through the outline, or a specified portion of it, for a given keyword or phrase and, optionally, replace it with another word or phrase.

Window - adjust the "window" of the screen which looks at the outline. The window can be moved in four different directions.

Porting

Port - move the outline, or part of it, to another disk file or to the printer. This is one of the most versatile and powerful commands in ThinkTank. If the outline, or a part of it is to be transferred to another outline, a disk file must be used as an intermediary.

Three different styles of porting are available. Structured, used primarily in disk porting, is used to backup an outline or transfer material from one outline to another. Plain, which uses minimal format settings, is used primarily to port the outline to a textfile for editing by a word processor. Formatted is used primarily for printing.

There are 16 separate format settings. These include:

1. Line spacing (single, double or triple);
2. Left margin (0-40);
3. Right margin (39-131);
4. Lines per page (20-10000);
5. Flat or indented headlines;
6. Amount for indentation (0-15);
7. Number of levels of headlines ported (0-10000);
8. Number of levels of paragraphs ported (0-10000);
9. Whether headlines are to be numbered;
10. Whether a table of contents is to be generated;
11. Number of levels of headlines included in the table of contents;
12. Whether page headers are printed;
13. Whether page footers are printed;
14. Whether single sheet or continuous feed paper is to be used;
15. Number of copies;
16. Printer initialization code, if any.

Copy - copy an outline heading or paragraph.

Alpha - sort, in alphabetical order, all headings immediately subordinate to a specified heading.

Promote - make subheadings equal in level to a specified heading.

Merge - combine two paragraphs into a single one.

Keys - reconfigure the action keys or the editor command keys. This permits you to customize these single letter commands in ThinkTank to your own liking.

Menu - add user created functions to the program through the specialist menu.

Easy to Use

ThinkTank is a very easy program to use. The menus, with optional command keys to bypass the menus, provide maximum flexibility. The command names on each menu are quite descriptive and when a selection is made, a more complete description of the command is given. Backing out of commands is easy -- in almost all cases all you need do is press the "Escape" key.

The program requires no prior expertise to use it. Although the program is written in Pascal for the UCSD p-System, the user does not need to know anything about the p-Sytem or Pascal to operate it. A special disk management menu is provided for formatting disks, copying them, and listing their directory. Indeed the special programs even refer to disk drives by their Apple DOS names of Drive 1 and Drive 2 rather than the more confusing p-System names of Volume #4 and Volume #5.

p-System Environment

The p-System environment does create a few problems, however, in using some of the more advanced features of ThinkTank. But these are problems only to a user unfamiliar with the p-System.

For example the text file which is created by the Port command is not an Apple DOS text file but a p-System text file. To edit it you must either use a p-System word processor, the p-System text editor, or transfer the file to an Apple DOS disk.

Several good programs exist for transfering files from the p-System to Apple DOS and from Apple DOS to the p-System. One of the best is the combination of HUFFIN and PUFFIN, originally published in *Call -Apple* and reprinted in *Call -Apple in Depth*, All About Pascal.

Superb Documentation

The quality of the documentation is superb. The manual is written in English, a language not usually found in computer documentation. The format also follows the ideal of an introduction, a short demonstration, a tutorial, and a reference guide. The manual even handles the problems of using different configurations of Apples in a coherent manner.

The publisher offer a series of technical notes available at nominal extra cost. These notes supply extra information on the use of a hard disk, the use of ThinkTank with word processors, the format of the data file, and the means of adding custom commands to ThinkTank's specialist menu. Additional techinical notes are planned.

ThinkTank is described as an "open system" with much opportunity for customization. This is accomplished in several ways.

1. The availability of the technical notes.

2. The ability to change the command keys.

3. The ability to add your own commands to the specialist menu.

4. The lack of any copy protection for the program.

This program is one of my favorites. Its concept is brilliant and its implementation is virtually flawless.

ThinkTank is available for $150.00 from:
Living Videotext, Inc.
450 San Antonio Road, Suite 56
Palo Alto, CA 94306

```
########################################
#                                      #
#          AUTHORS WANTED              #
#                                      #
#  Share your experiences, Apple       #
#  programming hints,pascal tips,      #
#  software reviews. Send your         #
#  material to the Cider Press         #
#  Editor at the Clubhouse.....        #
#                                      #
########################################
```

NUER'S

by Kathy Rudovsky

The November Nuer's Meeting had an enthusiastic group of Apple Users. Experience ranged from one month to two years and everything in between.

During the "Walk Through Your Computer" session, we strolled through the peripheral slots. We discovered 8 slots in the II+, cleverly numbered 0 to 7 and 7 slots in the IIe, not as cleverly numbered.

You can plug your peripherals into any slot you wish with a few exceptions, however, a uniform standard is not only preferred, but demanded by many software programs, CPM and the Pascal Language.

Slot 0 - 16k Ram Card (Language Card)
Slot 1 - Printer
Slot 2 - Modem
Slot 3 - 80 Column Card or External Terminal
*Slot 4 - Speech Synthesizers, music cards, clocks, graphics tablets
*Slot 5 -
Slot 6 - Disk Operating System (DOS)
*Slot 7 -

* = Optional Slots

During "The Software Session" we took a look at Word Processors including Bank Street Writer demonstrated by Ella Miyamoto, Magic Window demonstrated by Bob & Lita Warnock and Apple Writer II demonstrated by Kathy Rudovsky.

DID YOU KNOW? -

?SYNTAX ERROR = Applesoft error in Syntax

SYNTAX ERROR = DOS error in Syntax

If you are copying an entire disk with COPYA you need not initialize (format) the new diskette first. This procedure is automatic within the copy program.

NUER'S rendezvous on the third Wednesday of each month at 7:30 pm at the SFAC Clubhouse, 1515 Sloat Blvd. #2. Anyone interested or curious is wecome to come. The atmosphere is informal and friendly.

PRODUCT	MTG PRICE	MAIL PRICE
Library #1	$98.00	$101.50
Library #2	$98.00	$101.50
1981 Yearbook	$98.00	$101.50
COLLECTIONS		
Basic Tut Pak(3)	$20.00	$22.00
Forth Pak (3)	$20.00	$22.00
Game Set (2)	$15.00	$17.00
Home Apps(2)	$15.00	$17.00
Education App(2)	$15.00	$17.00
1983 Prog Util(1)	$10.00	$12.00
VisiCalc Temp1(1)	$10.00	$12.00
VisiCalc Util (1)	$18.00	$20.00
IAC BASE	$10.00	$12.00
Mailing List	$10.00	$12.00
SFAC Disk 1	$ 7.50	$ 9.50
SFAC 1983 Disk 2	$ 7.50	$ 9.50
SFAC 1983 Disk 3	$ 7.50	$ 9.50
SFAC 1983 Pasc 1	$ 7.50	$ 9.50
Mach Lang Util	$10.00	$12.00

All foreign orders add 25% to above prices.

SFAC MAIL ORDER BLANK

NAME.....

ADDRESS.....

CITY.....STATE.....

ZIP.....

MEMBER NUMBER:_____

INTEREST GROUP LEADERS

CP/M	Jim Ayers	383-0473(day) 383-5759(eves)
East Bay	Molly Mitchell	526-9541
Epson	Michael Fischer	557-1250 (day) 933-4435(eve)
Forth	Dudley Ackerman	626-6295
Green Apples (BASIC)	Gary Owens	521-4002
Marin Graphics	Reed Harris	924-7221
	Bob Sachs	332-3331
Nu'ers	Cathy Rudovsky	756-6480
Pascal	Mike Schnapp	386-2238
Product Review	"Woody" Liswood	585-6375
Telecommunications	Frank Fennelly	(408)984-7321
VisiCalc	Don Scellato	672-2051(eve)

SF APPLE CORE
1515 Sloat Blvd., Suite 2
San Francisco, CA. 94132

TIME VALUE

Bulk Rate
U.S. Postage
PAID
Permit #85
El Cerrito, CA 94530

TO:

3121AUG84
DOUGLAS STORKOVICH
921 PINTO CT
WALNUT CREEK CA 94596